MTL - 1

MICROPROCESSOR TRAINING LABORATORY

REFERENCE MANUAL

6700

Cleveland Institute of Electronics
1776 East 17th Street
Cleveland, Ohio 44114

MTL - 1

MICROPROCESSOR

TRAINING

LABORATORY

REFERENCE MANUAL

6700

Cleveland Institute of Electronics, Inc.
1776 East 17th Street
Cleveland, Ohio 44114

CONTENTS

I INTRODUCTION

THE CIE MTL-1 MICROPROCESSOR TRAINER ... Page 1
FCC COMPLIANCE AND RFI ... Page 2
ABOUT THIS REFERENCE MANUAL ... Page 3

II USE OF THE CIEBUG MONITOR

COMMAND STRUCTURE ... Page 4
KEYPAD DEFINITION ... Page 5

CIEBUG MONITOR COMMANDS

DATA Command ... Page 6
ADDR Command ... Page 6
GO Command ... Page 7
STEP Command ... Page 9
SET BREAKPOINT Command ... Page 10
MOVE Command ... Page 10
CLEAR BREAKPOINT Command ... Page 11
PC REG key - DP REG key ... Page 11
READ Command ... Page 12
WRITE Command ... Page 13

III MEMORY MAPS

TRAINER MEMORY MAP 2K SYSTEM ... Page 14
TRAINER MEMORY MAP 4K SYSTEM ... Page 16
SYSTEM DIRECT PAGE DEFINITIONS ... Page 18

IV CIEBUG UTILITY FUNCTIONS

BLOCK FILL FUNCTION ... Page 19
RELATIVE ADDRESS AID ... Page 20
GENERAL PURPOSE CALCULATOR ... Page 22
FIND BYTES FUNCTION ... Page 26

V CIEBUG ROM ROUTINE LIBRARY

ACCESSING ROUTINES ... Page 27
ROUTINE DESCRIPTIONS ... Page 28

VI TRAINER HARDWARE INFORMATION

SYSTEM BLOCK DIAGRAM ... Page 51
CONNECTORS ... Page 52
SIGNAL SPECIFICATIONS AND MAXIMUM LOADING ... Page 53
CABLE ... Page 53
SEMICONDUCTOR PARTS LIST ... Page 54
DISPLAYS ... Page 54
CIEBUG MONITOR DIAGNOSTICS ... Page 54
IN CASE OF TROUBLE ... Page 55
CASSETTE TAPE RECORDER GUIDELINES ... Page 55
CASSETTE SPEED PARAMETER ADJUSTMENT ... Page 56

VII 6809 MICROPROCESSOR REFERENCES

CLEVELAND INSTITUTE OF ELECTRONICS LESSONS ... Page 58
OTHER PUBLICATIONS ... Page 58

Copyright © 1984. Cleveland Institute of Electronics, Inc.
All Rights Reserved/Printed in the United States of America.

FIRST EDITION/Eleventh Printing/May 1998

I - INTRODUCTION

THE CIE MTL-1 MICROPROCESSOR TRAINER

The C.I.E. Microprocessor training laboratory, model MTL-1, is a complete microcomputer system designed to specifically help to teach microprocessor fundamentals (both hardware and software) in concert with CIE's exclusive Auto-Programmed (TM) lessons. Expandability was engineered into the trainer design to keep pace with the rapidly changing world of microprocessors. The advanced 6809 microprocessor provides a modern 16-bit/8-bit microprocessor design. The unique software and hardware design of the MTL-1 provides an effective learning laboratory for the beginning microprocessor student as well as a useful tool for the more advanced student.

The heart of the trainer is Motorola's powerful 6809 microprocessor, with its internal 16 bit data paths and extensive instruction set. Instructions are executed at a maximum rate of nearly 448,000 per second, regulated by the crystal controlled clock frequency of 895 KHZ. The trainer is supplied with 2048 (2K) of programmable memory (RAM), expandable on-board to 4K. An eight digit, seven segment LED display and a keypad containing 19 keys are monitored by the latest in LSI controllers, further enhancing the capabilities of the trainer. The internal power supply can service the needs of almost any circuit that can be assembled on the large built-in solderless breadboard area. A full complement of processor signals, suitably buffered, is brought out to the breadboard area permitting the assembly of complex circuits illustrating the typical applications of microprocessor hardware. Future expansion is provided for by bringing many of these same signals out to an expansion connector at the back of the trainer. An eight bit parallel port with two control lines is dedicated for use in experiments.

A software monitor program, CIEbug, completes the training laboratory software environment. This 4K byte ROM program controls basic operation of the trainer, allowing full access to the available memory and processor registers through 20 easy-to-use commands. Breakpoints and hardware single step circuitry simplify testing of machine language programs. A standard audio cassette recorder may be used to save programs and data using the built-in high-speed (approximately 150 bytes per second) cassette interface. Many utility functions are resident in CIEbug, including a relative address aid and a general purpose programmer's calculator. Many support routines for the keypad and display are also provided by CIEbug, making student programs shorter and simpler. A well defined interface to the monitor subroutines promotes the use of position independent code, an inherent feature of the 6809's architecture. Built-in diagnostic routines are called each time the trainer receives power, thus insuring the integrity of the trainer hardware.

FCC COMPLIANCE AND RFI

FEDERAL COMMUNICATIONS COMMISSION
RADIO FREQUENCE INTERFERENCE
STATEMENT

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

.... reorient the receiving antenna
.... relocate the trainer with respect to the receiver
.... move the trainer away from the receiver
.... plug the trainer into a different outlet so that trainer and receiver are on different branch circuits.

If necessary, the user should consult the manufacturer or an experienced radio/television technician for additional suggestions.

The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems." This booklet is available from the US Government Printing Office, Washington, D.C., 20402, Stock No. 004-000-00345-4

ABOUT THIS REFERENCE MANUAL

This manual is not an AUTO-PROGRAMMED lesson on the CIE MTL-1 Microprocessor Trainer Laboratory. It only intended to provide a convenient compendium of reference MTL-1 information for the independent study student taking a CIE career, industrial, or associate degree program course that includes lessons on the CIE microprossor trainer.

Section II, USE OF THE CIEBUG MONITOR, describes the keypad and keyboard command operation of the MTL-1 Microprocessor Trainer laboratory. It describes the operation of the monitor commands: how they are entered, what they do, and how a command entry can be executed or canceled (aborted). Both the novice and experienced MTL-1 user will find this section a convenient operating reference

Also included in this document are descriptions (memory maps) of the memory areas used by the CIEbug monitor program. These are presented in Section III, MEMORY MAPS. Memory maps are given for both 2K and 4K MTL-1 systems. Section III also includes a handy listing of the symbolic names and zero page locations of the CIEbug monitor subroutines.

Section IV, CIEBUG UTILITY FUNCTIONS, describes four programming aid routines that are contained in the CIEbug ROM.

Section V, CIEBUG ROM ROUTINE LIBRARY, describes many of the routines that are built into the CIEbug monitor ROM. This information is presented for the advanced 6809 programmer who wishes to use the CIEbug monitor ROM as a library of subroutines that can be called by the user's program. This information is not intended for beginning programmers and the novice is cautioned to first master elementary machine language programming fundamentals and principles before attempting to use the information in this section to any great extent.

TRAINER HARDWARE INFORMATION, Section VI, contains information on semiconductor chips, connectors, cables, and other hardware used in the MTL-1. Also included is a system block diagram. Signal and loading specs are given. Should trouble be encountered with the trainer or with cassette tape recorder operation, the reader should consult the information on diagnostics and cassette recorder parameter settings in this chapter.

Section VII, 6809 MICROPROCESSOR REFERENCES lists a few publications that contain information on the 6809, 6809 based systems and programming, microprocessor systems in general, and the Motorola 6800 microprocessor (the predecessor of the 6809). The list is not intended to be exhaustive (there is extensive literature published in this field) but only representative.

As in any technical publication, errors are bound to appear. CIE would appreciate being notified of any errors that you encounter so that they may be corrected in subsequent printings of this manual. Please send a short note describing errors to: Director of Instruction, Cleveland Institute of Electronics, 1776 East 17th Street, Cleveland, Ohio 44114.

II - USE OF THE CIEBUG MONITOR

COMMAND STRUCTURE

This part of this document describes the operation of the various commands available on your 6809 microprocessor trainer. Reading each comand description, you should look at what the command does (and does not) do, the format for entering, executing (and possibly escaping from) a command, and sample displays identifying that particular command. After you gain experience with the trainer, you will only need to consult this document occasionally for reference purposes or to refresh your memory.

The CIE microprocessor training laboratory monitor program, 'CIEbug', has a comprehensive set of commands that are directly accessible from the hex keypad. Most commands are requested by holding the shift key down while pressing another key. This combination will be referred to in this manual as SHIFT/COMMAND, where command is the legend that appears in reversed lettering on the keycaps. Only two monitor commands do not require using the shift key. They are the memory data display key (DATA) and the increment key (+).

Commands are structured so that explicit confirmation must be given before any major action is taken. This gives the operator the opportunity to review the parameters, and 'escape' from the command if desired. One of the command keys,SHIFT/GO, has been defined to act as the universal abort key. The SHIFT/GO has a dual function. Its role is determined by the context of the command. For all commands, pressing SHIFT/GO at any time will exit from the command without causing the requested action to take place. The plus key (+) is the command that usually confirms 'everything is ok', and causes the command to be carried out.

The only time the SHIFT/GO key does not possess the 'escape' quality is when the monitor is not performing any other command, and the CIEbug logo is displayed. CIEbug is said to be 'at monitor level' at that time. Only at that time does the SHIFT/GO key change to a valid command key, which may be used to begin execution of your machine language program. See the GO command description for a further explanation.

A RESET switch is located on the back panel of the trainer. Pressing RESET causes a reinitialization of the processor and a restart of CIEbug.

A switch labeled I-MON (non-maskable Interrupt to the MONitor) is located on the trainer next to the keypad. Pressing the I-MON switch while executing a user program causes trainer control to be returned to CIEbug.

KEYPAD DEFINITION

The C.I.E. microprocessor trainer keypad is arranged as in the following figure:

```
+----------+  +----------+  +----------+  +---------+
: (ACC. A) :  : (ACC. B) :  : (CC REG) :  :(DP REG) :
:    A     :  :    B     :  :    C     :  :   D     :
+----------+  +----------+  +----------+  +---------+

+----------+  +----------+  +----------+  +---------+
: ( X REG) :  : ( Y REG) :  : ( U REG) :  : ( READ ):
:    7     :  :    8     :  :    9     :  :    E    :
+----------+  +----------+  +----------+  +---------+

+----------+  +----------+  +----------+  +---------+
: (CLR BR) :  : (PC REG) :  : (SP REG) :  : (WRITE) :
:    4     :  :    5     :  :    6     :  :    F    :
+----------+  +----------+  +----------+  +---------+

+----------+  +----------+  +----------+  +----------+
: ( STEP ) :  : (SET BR) :  : ( MOVE ) :  : ( ADDR ) :
:    1     :  :    2     :  :    3     :  :   DATA   :
+----------+  +----------+  +----------+  +----------+

+----------+  +----------+        +----------+
:   (GO)   :  :   (-)    :        :          :
:    0     :  :    +     :        :  SHIFT   :
+----------+  +----------+        +----------+
```

Where functions (lettered white on black) are obtained by pressing the shift key and the function key at the same time.

CIEBUG MONITOR COMMANDS

DATA COMMAND

This command allows you to display and modify any location in the trainer's available memory. The display will indicate the address of the location in the four leftmost digits, and the data stored at that location in the two rightmost digits. The decimal points will be lit in the data field, indicating that any hex data keys you may press will be shifted into the memory location. If the location is not in read/write memory, the display will not change when hex data is entered.

The plus key (+) will advance to the next location, while the minus key (SHIFT/-) will display the contents of the previous location. If the SHIFT/ADDR keys are pressed, you will be allowed to enter a new memory address. (See ADDR description below.) Any other shifted key will exit from this commannd, by performing its indicated function. For example, if one of the register display keys is pressed, that register contents will be displayed immediately.

In summary:

Valid keys:	0-F, +, -, ADDR, DATA
Exit:	Any other shifted function key.

ADDR COMMAND (SHIFT/ADDR)

This command is essentially the same as the DATA command above, except that the decimal points will be lit in the address field, indicating that hex data will form a new address.

As with the DATA command, the plus key advances to the next location, while the minus key backs up to the previous location. The DATA key puts you in data entry mode, and the ADDR key allows you to enter address information again.

In Summary:

Valid Keys:	0-F, +, -, ADDR, DATA
Exit:	Any other shifted function key.

GO COMMAND (SHIFT/GO)

Begins execution at the indicated address. Execution will continue until either an RTS (or equivalent) instruction is executed or a software interrupt is encountered (breakpoint). The decimal points indicate the start address may be modifed, by entering hex digits. In order to actually begin executing a program, press any key other than GO (pressing the plus key is preferred). The GO key has a two fold purpose - to begin execution and to abort a command in progess. These two functions are distinguished by the current command state. Control can only be transferred to a machine language routine from the 'monitor level' display, (the CIEbug logo) so the function of the GO key is to call the 'GO' function at this level. Conversely, while executing any other command, beginning execution of a machine language routine would make little sense, so the function of the GO key changes to that of 'terminate current command,' and return to monitor level. It is recommended that the register contents be set up using register display modify commands prior to executing the GO command.

The GO command sets the processor registers to the values last displayed by the register display commands by means of a Return from Interrupt (RTI) instruction. This means that the stack pointer must point to an area of at least 14 bytes of RAM memory. If it does not, an error message will be displayed, and the routine will wait for any key before returning to monitor level.

Upon encountering a base level return instruction (RTS, PULS, PC, etc.) the following registers will be saved, and control returned to the monitor: A, B, CC, DP, X, Y, and U.

Upon encountering a SWI instruction (breakpoint), all processor registers will be saved, and the display will indicate the instruction that was about to be executed. At this time, you may examine and change register contents using the register keys, display and modify memory locations using the ADDR and DATA keys, and set or clear a breakpoint using the command keys SHIFT/SET BR and SHIFT/CLR BR. A block move command is also available in this mode (SHIFT/MOVE). The plus key resumes execution at the indicated location. Pressing the GO key at this time will return to the monitor, as will any other shifted command key.

If during the execution of the machine language program, monitor variables on page zero (specifically, $C0 to $FF) are altered, then on reentering the monitor a cold start will occur. However, certain variables in this area may be altered without ill effect. These are described in another section.

In Summary:

Vaild Keys: 0-F, +, ADDR, DATA, PC REG - DP REG, MOVE, SETBR, CLRBR

Exit: Invaild keys. GO key after encountering a breakpoint.

```
+   +--+   +--+   +  +      +--+   +--+   +--+
:      :      :   :  :         :   :      :
:   +--+   +--+   +--+         :   +--+   +--+
:   :         :      :         :      :   :  :
+   +--+   +--+      +      +--+   +--+   +--+
```

SAMPLE DISPLAY
INDICATING BREAKPOINT

STEP COMMAND (SHIFT/STEP)

Begins execution at the indicated address, for one instruction. Decimal points indicate the address may be changed prior to beginning. At this time, the GO key will return to the monitor, and the plus key will execute a single instruction. Again, it is recommended that register contents be set prior to beginning this step.

The same restrictions apply as for the GO command, namely, 14 bytes of RAM memory must exist below the stack pointer, and the display indicates the instruction about to be executed. While the instruction is being displayed, the register display, memory display, and block move commands are available. Breakpoints may be set or cleared while stepping. However, you may not step past the location at which a breakpoint is set. Either clear the breakpoint (using SHIFT/CLR BR) or set it someplace else (using SHIFT/SET BR). If you are stepping through a loop and wish to exit prematurely, simply set a breakpoint outside the loop and exit single step and begin GO at the current PC location. When you encounter the breakpoint, SHIFT/GO to get out of the GO command, clear the breakpoint, and resume single step. This provides a facility to skip through 'uninteresting' segments of code at full speed, and resume single step operation when the breakpoint is encountered.

In Summary:

Active Keys:	+, ADDR, DATA, PC REG - DP REG keys, MOVE, SET BR, CLR BR
Exit:	Invaild key. GO key preferred.

```
+   +--+   +--+   +  +     +--+   +--+   +--+
:      :      :   :  :            :      :
:   +--+   +--+   +--+     +--+   +--+   +--+
:   :         :      :               :   :  :
+   +--+   +--+      +     +--+   +--+   +--+
```

SAMPLING DISPLAY
DURING SINGLE STEP

SET BREAKPOINT (SHIFT/SET BR)

This command will replace a given instruction with a software interrupt instruction, which the monitor will interpret as a breakpoint in the program. The prompt and display indicate the address where a breakpoint will be placed. (Defaults are the current program counter, or the last addressed memory location.)

The breakpoint will be set only if any key other than GO is pressed (after entering a new address,for example.) The GO key will not change the breakpoint location. The entry is checked to be sure it is in RAM memory. If you attempt to place a breakpoint in non-RAM memory, a display will inform you of the error, and clear any previous breakpoint, after waiting for you to press any key.

In Summary:

Valid Keys:	0-F, GO.
Exit:	Invalid keys.

MOVE COMMAND (SHIFT/MOVE)

Often in the course of program development, one needs to shift a sequence of instructions around in memory to make room for more instructions, or to delete unused instructions. This command speeds this process. Upon entering this command, you will be prompted for the start address of the block to be moved (-SA on the display), the end address of the block (-EA), and the destination to which the block is to be moved. (-dA).

The decimal points indicate address information is being accepted. The plus and minus keys allow the review and changing of any of the three parameters. Plus advances to the next parameter, and minus backs up to the previous entry. When the information is correct, press any key except GO (+ preferred) while the destination address is displayed, and the block will be moved. Pressing the GO key, or any other key when the start or end addresses are displayed, will exit to the monitor with no move taking place. There is no restriction on the location of the blocks. But be aware that the monitor stack is located on the last page of available RAM and moves into this area can cause unpredictable results.

In Summary:

Valid Keys:	0-F, +, -, GO.
Exit:	As outlined above.

CLEAR BREAKPOINT (SHIFT/CLR BR)

This command serves two purposes. First, it 'undoes' what the set breakpoint command does, that is, removes the SWI instruction that was put in place of an instruction in the program. (Note: SWI does not actually replace the opcode until SHIFT/GO is given.) Second, it can be used to determine if and where a breakpoint has been set, without clearing it.

If no breakpoint is set, this command has no effect, and you will continue to see the monitor level display (CIEbug logo).

On the other hand, if a breakpoint is set, its location is displayed, with a 'clear break' prompt (-cbr). Pressing any key except GO will clear the breakpoint and return to monitor level, while the GO key will simply return to monitor level. This allows you to determine the location of a breakpoint without removing it.

In Summary:

Valid Keys: GO

Exit: Invalid keys (any key except GO)

PC REG KEY THROUGH DP REG KEY (SHIFT/PC REG THROUGH SHIFT/DP REG)

Pressing any of the above keys establishes the register display/modify mode. The decimal points indicate new data will be accepted, and the prompt indicates which register is currently being displayed. While in this mode, the plus key advances to the next register in the sequence (PC, SP, X, Y, U, A, B, CC, DP) with 'wrap around' on the last register. The minus key backs through the same sequence, again with wrap around. Hex data alters the contents of the registers, and any shifted function key exits.

Only the Condition Codes Register is displayed without a text prompt. It is, however, displayed as a series of eight ones and zeros, with a single decimal point lit in bit zero. The plus key moves the decimal point to higher-order bit positions, while the minus key moves to lower-order bits. You are expected to enter new data in the form of ones and zeros only. Any other entry will exit register display mode. Advancing the decimal point past bit seven will rejoin the above mentioned register sequence, as will backing up below bit zero.

Valid Keys: 0-F, +, -, other register keys

Exit: Invalid keys, GO preferred.

READ COMMAND (SHIFT/READ)

This command allows you to read a previously recorded tape into the trainer's memory. You are prompted for the file name (4 hex characters, of which only the least significant three are used), and any offset you may wish to add to the load address contained on tape.

When prompted for file name and offset, the + and SHIFT/- keys work as in the SHIFT/MOVE command. Any key except GO begins searching for the file, while pressing the GO key at any time exits the routine. (File name of FFF matches the next file on the tape). During the read process, the display will show the current file name being read from the tape, and the address where the data is being stored, if the file matches the specified name. If an error occurs, a dash will appear in digit four, and the start and end of the detected error will be stored in two page zero locations(TAPERS,TAPERE).

If the file name does not match, the read process will continue, but no address information will be displayed until the last byte, when the last location (including any offset) will be displayed, along with the error indication if an error was detected. The data will still be read from tape, and checked for parity and checksum errors. This provides a method for identifying and verifying a previously written tape. (Search for some non-existent file name).

During tape operations, the data read from the tape will be echoed to the trainer's internal speaker.

When a successful load is completed, the trainer will wait for you to press any key before returning to the monitor. You can distinguish this state by the absence of any sound from the internal speaker.

In Summary:

Valid Keys: 0-F, +, -, GO

Exit: Invalid keys

```
+    +--+   +--+          +  +   +--+   +--+   +--+
:       :      :          :  :   :      :         :
:    +--+   +--+   +--+   +--+   +--+   +--+      :
:    :         :             :      :   :  :      :
+    +--+   +--+             +   +--+   +--+      :
```

SAMPLE DISPLAY
DURING TAPE READ WITH ERROR
(FILE NAME IS 123, ADDRESS IS 4567)

WRITE COMMAND (SHIFT/WRITE)

This command allows you to prepare a tape from any data in the trainer's memory. You are prompted for file name, start address, and end address of the block of data to be written to tape. Pressing any key except GO will start the tape operation, so have the cassette hooked up and in record mode. As the data is being written to tape, it will be echoed to the internal speaker.

The display will indicate the file name, and the address of the data as it is being written.

In Summary:

Valid Keys: 0-F, +, -, GO.

Exit: Invalid Keys.

```
+  +--+  +--+     +  +  +--+  +--+  +--+
:     :     :     :  :  :     :        :
:  +--+  +--+     +--+  +--+  +--+     :
:  :        :        :     :  :  :     :
+  +--+  +--+        +  +--+  +--+     :
```

SAMPLE DISPLAY
DURING TAPE WRITE
(FILE NAME IS 123, ADDRESS IS 4567)

III - MEMORY MAPS

TRAINER MEMORY MAP 2K SYSTEM

(REVISION d.0)

```
0000 +----------------------------+
     :                            :
     :          STUDENT           :
     :      DIRECT PAGE AREA      :
00BF :                            :
     +----------------------------+
00C0 :  SPEAKER SILENCE FLAG      :
00C1 :  TAPE READ ERROR FLAG      :
00C2 :  TAPE READ ERROR START     :
00C4 :  TAPE READ ERROR END       :
00C6 :  TAPE FILE NAME            :
00C8 :  TAPE & SERIAL OFFSET      :
00CA :  TAPE FILE BLOCK START     :
00CC :  TAPE FILE BLOCK END       :
     +----------------------------+
00CE :  MONITOR TEMPORARIES       :
00CF :                            :
     +----------------------------+
00D0 :  TRANSIENT VARIABLES       :
     :  USED BY UTILITY ROUTINES  :
00D5 :                            :
     +----------------------------+
00D6 :  MONITOR RESET INTERLOCK   :
00D9 :                            :
     +----------------------------+
00DA :   REGISTER SAVE AREA       :
00E6 :   (IN 'RTI' ORDER)         :
     +----------------------------+
00E8 :  MONITOR RAM UPPER LIMIT   :
     +----------------------------+
00EA :  TAPE SPEED PARAMETER      :
     +----------------------------+
00EB :  BREAKPOINT INFORMATION    :
00ED :                            :
     +----------------------------+
00EE :  MEMORY DISPLAY            :
     +----------------------------+
00F0 :  MONITOR TEMPORARY         :
     +----------------------------+
00F2 :  INTERRUPT VECTOR - SWI3   :
00F4 :  SWI2 VECTOR               :
00F6 :  FIRQ VECTOR               :
00F8 :  IRQ  VECTOR               :
00FA :  SWI  VECTOR BREAKPOINT    :
00FC :  NMI  VECTOR STEP/I-MON    :
00FE :  RESET VECTOR              :
     +----------------------------+
     +-----------------------------+
0100 :   AVAILABLE STUDENT         :
     :   PROGRAM AND DATA          :
     :   MEMORY                    :
     :                             :
     :                             :
     +-----------------------------+
06FF :  START OF STUDENT STACK     :
     +-----------------------------+
0700 :                             :
     :  MONITOR RESERVED STACK     :
     :         AREA                :
     :                             :
     :                             :
07FF :  TOP OF MONITOR STACK       :
     +-----------------------------+
0800 :  ON BOARD EXPANSION RAM     :
0FFF :       (IC4)                 :
     +-----------------------------+
1000 :                             :
     :  EXPANSION MEMORY AREA      :
7FFF :                             :
     +-----------------------------+
8000 :  PIA PORT A DATA SYSTEM     :
8001 :  PIA PORT A CONTROL         :
     +-----------------------------+
8002 :  PIA PORT B DATA STUDENT    :
8003 :  PIA PORT B CONTROL         :
     +-----------------------------+
8004 :  IMAGE OF $8000-$8003       :
8FFF :                             :
     +-----------------------------+
9000 :  KEY/DISPLAY DATA           :
9001 :  KEY/DISPLAY CONTROL        :
     +-----------------------------+
9002 :  IMAGE OF $9000-$9001       :
9FFF :                             :
     +-----------------------------+
A000 :  I/O EXPANSION AREA         :
     :  (AND EXPERIMENTS)          :
BFFF :                             :
     +-----------------------------+
C000 :  IMAGE OF $F000-$FFFF       :
EFFF :                             :
     +-----------------------------+
F000 :  C.I.E. TRAINER MONITOR     :
     :       (CIEBUG ROM)          :
FFEF :                             :
     +-----------------------------+
FFF0 :  HARDWARE INTERRUPT         :
FFFF :            VETORS           :
     +-----------------------------+
```

TRAINER MEMORY MAP 4K SYSTEM

(REVISION d.0)

```
0000 +---------------------------+
     :                           :
     :          STUDENT          :
     :      DIRECT PAGE AREA     :
00BF :                           :
     +---------------------------+
00C0 :  SPEAKER SILENCE FLAG     :
00C1 :  TAPE READ ERROR FLAG     :
00C2 :  TAPE READ ERROR START    :
00C4 :  TAPE READ ERROR END      :
00C6 :  TAPE FILE NAME           :
00C8 :  TAPE & SERIAL OFFSET     :
00CA :  TAPE FILE BLOCK START    :
00CC :  TAPE FILE BLOCK END      :
     +---------------------------+
00CE :  MONITOR TEMPORARIES      :
00CF :                           :
     +---------------------------+
00D0 :  TRANSIENT VARIABLES      :
     :  USED BY UTILITY ROUTINES :
00D5 :                           :
     +---------------------------+
00D6 :  MONITOR RESET INTERLOCK  :
00D9 :                           :
     +---------------------------+
00DA :   REGISTER SAVE AREA      :
00E6 :   (IN 'RTI' ORDER)        :
     +---------------------------+
00E8 :  MONITOR RAM UPPER LIMIT  :
     +---------------------------+
00EA :  TAPE SPEED PARAMETER     :
     +---------------------------+
00EB :  BREAKPOINT INFORMATION   :
00ED :                           :
     +---------------------------+
00EE :  MEMORY DISPLAY           :
     +---------------------------+
00F0 :  MONITOR TEMPORARY        :
     +---------------------------+
00F2 :  INTERRUPT VECTOR - SWI3  :
00F4 :  SWI2 VECTOR              :
00F6 :  FIRQ VECTOR              :
00F8 :  IRQ  VECTOR              :
00FA :  SWI  VECTOR BREAKPOINT   :
00FC :  NMI  VECTOR STEP/I-MON   :
00FE :  RESET VECTOR             :
     +---------------------------+
     +----------------------------+
0100 :   AVAILABLE STUDENT        :
     :   PROGRAM AND DATA         :
     :   MEMORY                   :
     :                            :
     :                            :
     +----------------------------+
0EFF :  START OF STUDENT STACK    :
     +----------------------------+
0F00 :                            :
     :  MONITOR RESERVED STACK    :
     :        AREA                :
     :                            :
     :                            :
0FFF :  TOP OF MONITOR STACK      :
     +----------------------------+
1000 :                            :
     :  EXPANSION MEMORY AREA     :
7FFF :                            :
     +----------------------------+
8000 :  PIA PORT A DATA SYSTEM    :
8001 :  PIA PORT A CONTROL        :
     +----------------------------+
8002 :  PIA PORT B DATA STUDENT   :
8003 :  PIA PORT B CONTROL        :
     +----------------------------+
8004 :  IMAGE OF $8000-$8003      :
8FFF :                            :
     +----------------------------+
9000 :  KEY/DISPLAY DATA          :
9001 :  KEY/DISPLAY CONTROL       :
     +----------------------------+
9002 :  IMAGE OF $9000-$9001      :
9FFF :                            :
     +----------------------------+
A000 :  I/O EXPANSION AREA        :
     :  (AND EXPERIMENTS)         :
BFFF :                            :
     +----------------------------+
C000 :  IMAGE OF $F000-$FFFF      :
EFFF :                            :
     +----------------------------+
F000 :  C.I.E. TRAINER MONITOR    :
     :       (CIEBUG ROM)         :
FFEF :                            :
     +----------------------------+
FFF0 :  HARDWARE INTERRUPT        :
FFFF :          VETORS            :
     +----------------------------+
```

SYSTEM DIRECT PAGE DEFINITIONS

Symbolic names are used for variables in the monitor subroutine descriptions. Their locations in the monitor direct page (page zero) are given below. The contents of most locations are changed by various monitor commands and the contents depend on previously executed commands. Variable length is indicated by the number of bytes in the length column. The table below is for both a 2K and 4K trainer. Systems with 4K of RAM use the same locations but some contents are different.

Symbol	Location	Length	Comments
SQUAWK	C0	1	speaker on/off flag
TAPERR	C1	1	set if tape read error
TAPERS	C2	2	start address in memory where tape error occured
TAPERE	C4	2	end address of tape error
FNAME	C6	2	tape file name
OFFSET	C8	2	load address offset for tape and serial download
LOADA	CA	2	tape file start address
ENDA	CC	2	tape file end address
DSTORE	CE	1	flag used by cassette
VARACKS	CF	1	internal variable
BITCON	D0	2	RS 232 bit delay
HSTACK	D0	2	calculator stack pointer
BASE	D2	1	calculator radix
SMODE	D3	1	calculator display mode
OVFL	D4	1	arithmetic overflow
KEY	D6	4	monitor reset interlock
CCSAVE	DA	1	condition code register
AASAVE	DB	1	accumulator A
ABSAVE	DC	1	accumulator B
DPSAVE	DD	1	direct page register
IXSAVE	DE	2	index register X
IYSAVE	E0	2	index register Y
USSAVE	E2	2	user stack pointer
PCSAVE	E4	2	program counter
HSSAVE	E6	2	hardware stack pointer
RAMTOP	E8	2	monitor RAM limit
TAPESP	EA	1	tape speed parameter
BRKDAT	EB	1	data at breakpoint
BRKADR	EC	2	address of breakpont
POINT	EE	2	memory display pointer
MONSTK	F0	2	monitor stack pointer
SWI3V	F2	2	SWI3 vector
SWI2V	F4	2	SWI2 vector
FIRQV	F6	2	FIRQ vector
IRQV	F8	2	IRQ vector
SWIV	FA	2	SWI vector (breakpoint)
NMIV	FC	2	NMI vector (single step)
RESETV	FE	2	monitor reset

IV - CIEBUG UTILITY FUNCTIONS

BLOCK FILL FUNCTION (SWI3, $17)

Using the memory display/modify function of your C.I.E. 6809 microprocessor trainer, enter the following data starting at location 0000 (hex):

```
0000    11    SWI3
0001    3F
0002    17    FCB  $17
0003    39    RTS
```

Then using the GO command, execute the above code starting at 0000. All this is intended to do is to "call" the block fill subroutine that has been built into the monitor, and to provide some method for returning to the monitor when done. The above code could be replaced with a jump subroutine/return sequence using the absolute location of the routine, but the method illustrated above frees you from having to find that location in future versions of the monitor.

Once the block fill routine is entered, you will be prompted for the data to be used to fill the memory block, and the starting and ending locations defining the block itself. As with similar input situations in the monitor, the SHIFT/GO key sequence will "escape" without any further action; while the plus key advances to the next input, and the minus key (SHIFT/-) backs up to the previous entry. Note the data which will fill the defined block consists of 16 bits (or two bytes). If you wish to fill an entire area with, say 00, for example, make sure you enter the same data for both bytes. Also note that the block is defined as from the start address (-SA on the display) to the end addres (-EA on the display) inclusive. If the area defined contains an odd number of bytes, only the most significant byte of the data pattern will be stored in the last location. No data will be stored if the starting address is higher than the ending address.

A note of caution is in order here. While this routine can be very useful in clearing out a block of memory prior to its use in a program, no checking is done by the routine as to the location of the block. If you fill the monitor direct page locations ($C0-$FF) or the stack area (on the last page of RAM), the result may be a trainer that is "hung up". Reset should recover, though. Also note that this routine is very fast and very unforgiving. It is easy to wipe out many hours of work with a few keystrokes. Take the time to double check your entries before the final plus key. If there is ever any doubt, the GO key will return the monitor without filling anything.

RELATIVE ADDRESS AID (SWI3, $18)

This special function can be used to simplify the calculation and entry of the one and two byte two's complement relative offsets encouraged by the 6809 instruction set. This subroutine is intended to be used while a program is being entered into the trainer's memory, as follows:

1) Using the memory display/modify command, set up the following subroutine somewhere in RAM memory that is otherwise unused:

```
XX00  11     SWI3
XX01  3F
XX02  18                call relative address aid
XX03  39     RTS        return to CIEbug
```

2) Set the program counter register to XX00, the location at which the above subroutine was entered.

3) Begin entering the desired operation codes for the program you are working on.

4) When you need a two's complement offset in your program, position the memory display at the location where you would like the offset deposited, press SHIFT/GO once to leave display/modify memory, press SHIFT/GO and + to call up the relative address aid.

5) The relative address aid will prompt you for the size of the offset in bytes. Only one or two byte offsets are allowed. You will next be prompted for the absolute location the offset is to reference. Pressing + will calculate the required offset, and if it is within the size requested, deposit the value into memory at the place you last examined. Pressing SHIFT/GO will escape without calculating or changing anything.

   If the required offset is too large for one byte when a single byte offset is selected, or the offset is small enough to fit in one byte when two had been requested, a graphic display will fill the display and the speaker will beep several times. The memory display function of the monitor will be backed up to point at the OPCODE, so that may be corrected and a new offset requested.

6) If the offset is correctly specified, it will be deposited into memory as indicated above, and the memory display function will now point to the next available byte, so that you may continue entering your program.

The following example should help clarify the use and operation of the RELATIVE ADDRESS AID.

Enter	Comment
SHIFT/ADD, 0100, DATA	start of program
30, +, 1F, +	LEAX -1,X instruction
26, +	BNE Opcode, now need offset to reach 0100 note leave display point at place offset will be deposited (0103)
SHIFT/GO,SHIFT/GO,+	call up relative address aid (assumes subroutine stored in memory and PC properly set)
1, +, 0100, +	one byte offset, to 0100
	back in in CIEbug
DATA, 39	note left pointing past offset

The result of this sequence was to enter the following program:

```
0100  30 1F    LEAX -1,X
0102  26 FC    BNE $0100
0104  39       RTS
```

Notice that the offset ($FC) was automatically computed and placed into memory by the relative address aid.

GENERAL PURPOSE CALCULATOR

The C.I.E. 6809 microprocessor trainer is equipped with a general purpose calculator designed for educational use. It supports the conventional four arithmetic functions, plus a variety of special functions applicable to microprocessor programming. Entry (and display) of numbers can be in base 8, base 10, or base 16, and expressions are entered just as they would be written, what some calculator manufacturers call algebraic entry. Parentheses may be used to override the built-in hierarchy which governs evaluation. All internal operations take place on 16 bit signed two's complement numbers, though results may be displayed in an optional unsigned mode.

For the general purpose calculator functions the C.I.E. microprocessor trainer keypad is arranged as in the following figure:

```
+----------+   +----------+   +----------+   +----------+
:   AND    :   :    OR    :   :   EXOR   :   :  ADD +   :
:    A     :   :    B     :   :    C     :   :    D     :
+----------+   +----------+   +----------+   +----------+

+----------+   +----------+   +----------+   +----------+
:    (     :   :   HEX    :   :    )     :   :  SUB -   :
:    7     :   :    8     :   :    9     :   :    E     :
+----------+   +----------+   +----------+   +----------+

+----------+   +----------+   +----------+   +----------+
:  OCTAL   :   :   DEC    :   :          :   :  MUL *   :
:    4     :   :    5     :   :    6     :   :    F     :
+----------+   +----------+   +----------+   +----------+

+----------+   +----------+   +----------+   +----------+
:   NOT    :   :   MODE   :   :   CHS    :   :  MOD     :
:    1     :   :    2     :   :    3     :   :  DIV /   :
+----------+   +----------+   +----------+   +----------+

+----------+   +----------+          +----------+
:    GO    :   :    CE    :          :          :
:    0     :   :  EQUALS  :          :  SHIFT   :
+----------+   +----------+          +----------+
```

Where the upper legends are obtained by pressing the shift key and that function key at the same time.

One uses the General Purpose Calculator program by running it via the SHIFT/GO command of the CIEbug monitor. A convenient method is to use the monitor SWI3 service routine, with a call number of $19. ($11, $3F, $19, $39) When the calculator is first entered, a single zero is displayed in the rightmost digit of the display, indicating it is ready to accept numeric input. The default input (and output) number base is decimal, and the display mode is set to signed. Since you are probably acquainted with a conventional calculator, the following descriptions should serve to explain typical operation of the calculator. A few examples are included to illustrate some of the features.

First of all, when an entered expression is evaluated, the calculator operations are performed in accordance to a hierarchy precedence order as follows:

1) Unary Minus/Plus, Parenthesis, and Functions,
2) Multiplication, Division, and Modulus
3) Addition and Subtraction
4) Logical NOT,
5) Logical AND,
6) Logical OR and Exclusive OR

The operations above are listed in order of precedence from highest to lowest level according to priority. When evaluating a keyed in expression the operations will be performed in order of their priority level. Where operations have the same level they are performed in the order in which they are entered.

Memory Requirements:

Upon entry, a calculator "parameter stack" is established 128 bytes below the entry hardware stack pointer. The parameter stack grows downward as operations are left pending, either through normal order of operations, or from the use of parentheses. At the same time, the hardware stack pointer is moving down as operators are entered. It is possible to have the hardware stack overwrite part of the parameter stack in an excessively complex expression. While this causes the displayed result to be in error, no other damage will occur, since both stacks are reset when the equals key is pressed. To avoid this potential problem, break expressions having more than about 10 levels of open parentheses into sub-expressions, and use the chain calculation capability to continue.

Arithmetic Operations (+,-,*,/,MOD)

These binary operators require two arguments, and produce a single result. Only the modulus function is not normally available on a conventional calulator. Since all operations are performed on 16 bit signed integers, the division operation requires some way to handle the remainder. The MOD key acts just like the division key, except that the result of the operation will be just the remainder. The quotient is discarded.

Logical Operators (AND, OR, EXOR, NOT)

The first three of these Boolean operators (AND, OR, EXOR) are binary in the sense that they take two 16 bit numbers, and perform the appropriate bit-wise logical function, returning a single 16 bit result. The logical NOT operator is unary, that is it will operate on one number at a time, and simply complements the following quantity.

Operation Control Functions:

Input/Output Base (OCTAL, DECIMAL, and HEX)

These keys change the base of both the input and output formats. You may not change the base in the middle of a calculation entry. You may, however, display a result in any of three different bases. It is recommended that you select the input base at the beginning of a calculation, and use that base throughout the calculation to avoid confusion. Then when the result is displayed, change the base to whatever is desired.

Signed Display Control (MODE, CHS)

The MODE function is an alternate action function that changes the form in which results are displayed. Note however, tha internal operations are always carried out as 16 bit, signed two's complement functions, regardless of the display mode.

The CHS key is a simple way to change the sign of a number while entering it. It also has an alternate action, and is preferred over unary minus for number entry.

A provision is made for clearing a mistaken entry, the CE key. This key only functions when numeric input is being processed. It also clears any overflows that may have occured. The GO key (SHIFT/GO) serves as the "quit" key by returning to the CIEbug monitor

The EQUALS key completes all pending operations, and supplies as many right parenthesis as are necessary to complete evaluation of the expression. After pressing EQUALS, you may use the displayed result as the first operand of a new calculation (this is called chain calculation) or enter a new number to take its place.

It is recommended that the unary minus (and redundant unary plus) only be used with parenthetical expressions, and not with numeric entries. The CHS key is far more convenient in that case.

General Purpose Calculator Eamples:

In base ten, signed display mode:

1+2*3-4*5 AND NOT 7 = -16

1+2*3-4*5 OR NOT 7 = -5

Press CHS, MODE, and any of the other base keys (DEC, OCTAL) and observe the result.

In base sixteen (HEX), signed display mode:

$3E8 / $0A = $64

$3FF CHS MOD $64 = -$17

In base eight, unsigned display mode:

NOT @3 AND NOT @4 = @177770

@1 CHS = @177777

FIND BYTES FUNCTION (SWI3, $1A)

In order to find a two byte pattern occurring anywhere in memory, use the same procedure as was outlined in the block fill function to set up a call to the routine built into the monitor. However, use the following sequence:

```
0000    11    SWI3
0001    3F
0002    1A    FCB    $1A
0003    39    RTS
```

Upon entering the routine, you will be prompted for two "masks" and the bounds of the memory block you would like to search. As with other multiple inputs, plus key advances, minus (SHIFT/-) backs up, and 'GO' at any time exits.

The operations performed by the routine are: "EXCLUSIVE OR" the contents of each byte of memory with the first mask specified, then "AND" the result with the second mask. If the overall result is zero, a match is declared and the routine exits. Otherwise each byte in the specified area is tested in the same way. No change is made to any of the data in the memory block itself.

In general, you may consider the "EOR" mask to be the pattern you are looking for, and the "AND" mask as a way of specifying "Don't Cares". For example, if the EOR mask is some specific two byte pattern and the AND mask = $FFFF then there will be a match declared on encountering the specific two byte pattern, and that pattern only. If the EOR mask was the same, but the AND mask was $FF00, a match would be declared upon matching only the first byte of the pattern.

If the pattern was found, you will hear a several tones from the speaker before the monitor display reappears. The next time you examine memory, you will be observing the first byte of that pattern, as found by the routine. If no match occurred, your memory display will be pointing where you left it, and only a single tone will occur before the CIEbug monitor display pops up.

V - CIEBUG ROM ROUTINE LIBRARY

ACCESSING ROUTINES

Each routine's absolute address within the CIEbug ROM is held in a table starting at hexadecimal location $F000 (all numbers in this manual are decimal, unless prefixed by a '$', in which case they are hexadecimal.) You should always access any of these routines by using either the extended indirect addressing modes of the 6809 processor, or by the built-in monitor call interrupt, SWI3, that CIEbug sets up for you. You should not use the routine's absolute address in any of your code. This is to enable your programs to work with later revisions of CIEbug. The routine names are suggested mnemonics, and are not recognized by the trainer directly.

The two methods of accessing ROM routines can be demonstrated and clarified by the following example. Suppose that the programmer wishes to clear the display. The CIEbug ROM routine to clear the display resides in memory beginning at an absolute address that is contained in the table at $F008. The mnemonic name is DSPCLR and it's table call number is $04.

A program segment to clear the display using the monitor call interrupt, SWI3, is the three bytes of code:

```
11              SWI3
3F
04              call number
```

The monitor call interrupt uses the call number of the routine rather than an indirect address. The monitor expects the byte following the two byte SWI3 opcode (11,3F) to be the table call number. The above three bytes of code will clear the trainer display.

A program segment that uses the extended indirect method to implement DSPCLR is four bytes of code that implements JSR ($F008):

```
AD              JSR extended indirect
9F              postbyte
F0              address
08
```

One advantage of the monitor call interrupt is that it takes one less byte (3 vs 4 for the extended indirect method). You may use this method as long as your routines do not change the SWI3 vector on page zero ($00F2). For convienience, both the monitor call number and table entry are given in the routine descriptions in this chapter.

ROUTINE DESCRIPTIONS

Approximately sixty routines are described below. Each description contains a descriptive mnemonic name, a call number and a table address. Each description tells what each routine does, what values the routine expects in registers (Entry), and what values the routine may return (Exit). Also shown are flags and registers that may be effected by the routine (Alters) as well as the useage of memory locations (Parameters). Error conditions are also described, if applicable, and general comments and cautions are are included where appropriate.

Routine: COLD
Call Number: $00
Address: ($F000)

Entry: -
Exit: none
Alters: -

This is the 'cold' entry to CIEbug. Calling this routine has the same effect as turning off the power to the trainer, then back on again. Note that this routine is not a subroutine, and will not return to the calling program.

Routine: WARM
Call Number: $01
Address: ($F002)

Entry: -
Exit: none
Alters: -
Parameters: (RAMTOP) - monitor stack pointer
(RESETV) - place to jump to

This is the 'warm' entry to CIEbug. This routine may be used to circumvent some of CIEbug's internal power up tests. The net result of this routine is to set the monitor stack pointer to the value contained in the zero page variable RAMTOP, and perform an indirect jump to the location pointed to by the zero page variable RESETV, with direct page register equal to zero, and the IRQ and FIRQ interrupts masked. The code pointed to by (RESETV) must begin with the opcode for a branch always, BRA, instruction. It is not recommended that this routine be used for general re-entry to CIEbug. Note that this routine is not a subroutine, and will not return to the calling program.

Routine: INITPI
Call Number: $02
Address: ($F004)

Entry: -
Parameters: (SQUAWK) = $00 speaker off
= $FF speaker on
Exit: X = Base address of system PIA (IC5)
Alters: CC,A,X

Calling this subroutine initializes the system PIA to the configuration necessary for proper operation of CIEbug. Since the speaker is controlled through this PIA, a software switch direct page location ,SQUAWK, has been made available to silence the speaker. If SQUAWK contains zero when this subroutine is called, speaker output will be disabled. If SQUAWK is set to $FF before calling this subroutine then speaker output will be enabled. This subroutine may also be used to obtain the beginning address of the system PIA.

doutine: INITKD
Call Number: $03
Address: ($F006)

Entry: -
Exit: X = Base address of system Keyboard/Display controller (IC6)
Alters: CC,A,X

Calling this subroutine initializes the system keyboard/display controller to the configuration necessary for proper operation of CIEbug. This routine also clears the display, and may be used to obtain the beginning address of the keyboard/display controller.

Routine: DSPCLR
Call numberer: $04
Address: ($F008)

Entry: -
Exit: -
Alters: none

This subroutine clears the seven segment display.

Routine: DSPHX2
Call Number: $05
Address: ($F00A)

Entry: A = number to display as two HEX digital
C bit in CC = 1 if number to be displayed with decimal points,
C bit in CC = 0 if number is to be displayed without decimal points
B = digit number of LSD on display.

Exit: B = B + 2 (MOD 8)
Alters: B

Subroutine DSPHX2 may be used to display two HEX digits anywhere on the display. Accumulator A contains the number to be displayed, while accumulator B specifies where to begin the display. Digit 0 is the rightmost digit of the display, while digit 7 corresponds to the leftmost display. Notice that register B is returned ready for display of two more digits to the LEFT of those just displayed. The display will wrap around from digit 7 to digit 0.

Routine: DSPHX4
Call Number: $06
Address: ($F00C)

Entry: X = number to display as four HEX digits
C bit in CC = 1 if number to be displayed with decimals points,
C bit in CC = 0 if number is to be displayed without decimal points
B = digit number of LSD on display.

Exit: B = B + 4 (MOD 8)
Alters: B

Subroutine DSPHX4 may be used to display four HEX digits anywhere on the display , in much the same manner as routine DSPHX2. Register X contains the number to be displayed, while accumulator B specifies where to begin the display. Digit 0 is the rightmost digit of the display, and digit 7 is the leftmost display. Notice that accumulator B is returned ready for display of four more digits to the LEFT of those just displayed. The display will wrap around from digit 7 to digit 0.

Routine: DSPHEX
Call Number: $07
Address: ($F00E)

Entry: A = number to display as one HEX digit (least significant nibble only)
C bit in CC = 1 if number to be displayed with decimal points,
C bit in CC = 0 if number is to be displayed without decimal points
B = digit number on display.

Exit: B = B + 1 (MOD 8)
Alters: B

This subroutine converts the least significant nibble of accumulator A into an appropriate pattern, and displays that pattern at the digit specified by accumulator B. Accumulator A is returned unchanged, and accumulator B is incremented by one (MOD 8), so that successive calls to this routine will display digits from right to left. Digit numbers for the display remain as defined for DSPHX2 and DSPHX4.

Routine: DSPMSG
Call Number: $08
Address: ($F010)

Entry: X = pointer to display string in memory,
(in left to right order)
B = Digit number to start in LSN,
digit number to stop in MSN
Exit: X = X + (STOP-START) + 1
Alters: X

DSPMSG is used to display a sequence of patterns on the seven segment display which are not necessarily numbers. An example is the frequent 'text' prompts which CIEbug issues. Register X points to an area in memory where patterns are stored as a series of single byte entries. Each pattern will be written to the display with no modification, from right (start) to left (stop). Accumulator B is used to specify where to begin and end the display, and hence its length. This information is encoded into the low and high nibbles, respectively, of register B. Each bit in the pattern, if set, will light a particular display segment. Bit definitions are: bit 7 = dp, bits 6 through 0 are segments g through a in order. The following example may clarify this routine a bit. It displays the 'CIEbug' message in the center of the display, then waits for any key to be pressed on the keyboard before returning.

```
        LEAX CIEMSG,PCR    make X point at string
        LDB #$61           stop digit is 6, start at 1
        SWI3
        FCB $08            call DSPMSG routine
        SWI3
        FCB $0D            (call WAIKEY)
        RTS                return to CIEbug
CIEMSG  FCB $B9            pattern for "C."
        FCB $86            pattern for "I."
        FCB #F9            pattern for "E."
        FCB $7C            pattern for "b"
        FCB $3E            pattern for "u"
        FCB $6F            pattern for "g"
```

Routine: DSPPTN
Call Number: $09
Address: ($F012)

Entry: A = pattern to be displayed
B = digit number on display
Exit: B = B + 1 (MOD 8)
Alters: B, CC(N,Z,V)

The display pattern subroutine simply displays whatever pattern is contained in accumulator A on the display at the digit specified in accumulator B. Definition of the pattern is the same as for routine DSPMSG, (bit 7=dp, bits 6-0 are segments g - a), while the digit number remains as it was for DSPHEX, DSPHX2 and DSPHX4.

Routine: DSPALL
Call Number: $0A
Address: ($F014)

Entry: A = pattern to display in all digits
Exit -
Alters: none

Calling this routine will fill the display with the patten contained in accumulator A. All digits of the display will be filled with the specified pattern. For example, to display a row of dashes, call this routine with accumulator A containing $40 (segment g).

Routine: DSPMS8
Call Number: $0B
Address: ($F016)

Entry: R(X) - pointer to 8 byte string in memory in MSD to LSD order.
Exit: R(X) = R(X) + 8
Alters: R(X)

This subroutine may be used to display a sequence of eight patterns on the seven segment display. The pattern in the memory location pointed to by register X on entry will be displayed in the left-most display digit, with the byte at R(X) + 1 in the next digit to the right, and so on. Note that R(X) will point to the byte just past the display string.

Routine: KEYPOL
Call Number: $0C
Address: ($F018)

Entry: -
Exit: R(CC) Z = 1 (BEQ) if no key present in FIFO
R(A) = 0;
R(CC) Z = 0 (BNE) if key present in FIFO,
R(A) = key code
Alters: CC, A

KEYPOL may be used to see if a keyboard key has been pressed. If no key was pressed, accumulator A will return clear and the Z flag will be set. If a key was pressed, the Z flag will be clear and the key code will be placed in accumulator A. The debounce of the keyboard is performed by the key/display controller, IC6. Only one entry will be placed in the internal FIFO for each key depression/release. This routine may not be used to detemine how long a key is pressed, only that one was pressed and released. Key codes are as labeled on the keycaps, with key 0 having a code of $00, key 1 returning $01, and so on through key F which has a code of $0F. The + key is encoded as $10, and the DATA key returns a $11. Pressing the shift key at the same time as another key will add $80 to the key code. This means that the - key is encoded $90 and the address key is $91. Note, even if the key code is $00, the Z flag will return clear if a key is detected.

Routine: WAIKEY
Call Number: $0D
Address: ($F01A)

Entry: -
Exit: after key pressed
R(A) = key code
Alters: CC,A

This subroutine will wait until any key is pressed on the keyboard, at which time the key will be acknowledged by 50 mS of a 440 HZ tone on the speaker. Keycodes remain the same as defined for routine KEYPOL.

Routine: BEEPLO
Call Number: $0E
Address: ($F01C)

Entry: -
Exit: after 50 mS of 330 HZ
Alters: none

Subroutine BEEPLO may be used to sound a short tone over the speaker. The monitor uses this routine to provide audible feedback that a key has been pressed.

Routine: BEEPHI
Call Number: $0F
Address: ($F01E)

Entry: -
Exit: after 50 mS of 880 Hz
Alters: none

Subroutine BEEPHI may be used to sound a short, high frequency tone over the speaker. The monitor uses this routine to warn of various error conditions.

Routine: TONE
Call number: $10
Address: ($F020)

Entry: R(A) = one cycle period in 10 uS
R(B) = duration in cycles
Exit: after (B) cycles of (A)
Alters: CC

You may use the TONE subroutine to generate various sound effects via the internal speaker. The useful range of period is from 3 (about 33KHZ) to 255 (390HZ). A cycle count of zero (R(B)) is interpreted as 256 cycles.

Routine: INPHEX
Call Number: $11
Address: ($F022)

Entry: R(U) = default value
R(CC) C bit = 1 if two digit hex input,
R(CC) C bit = 0 if four digit hex input
Exit: R(U) = new value
R(A) = last non-hex (or shifted) key pressed.
Alters: A,U

INPHEX is a general purpose hex input routine. On entry, R(U) contains a 'default' value (the entire register if four digit input, or just the low half if a two digit input is requested). This default value will be displayed starting at digit 4 of the display, with decimal points on. Pressing hex keys on the keyboard will shift the value left one digit, and place the value of that hex key in the LSD. Pressing any non-hex key (+/-, ADDR/DATA) will terminate entry, and the code of that key will be placed in the returned accumulator A. If no hex keys are pressed, the default value will be returned in R(U), otherwise the value will reflect the last displayed number. The high half of R(U) is ignored if two digit input is specified, however it may be altered upon return.

Routine: INPSEQ
Call Number: $12
Address: ($F024)

Entry: R(X) = pointer to start of prompts, each 4 bytes long in sequential order.
R(B) = total number of inputs desired
stack containing R(B) default values.
Exit: R(B) = actual number of entries
R(A) = last non-hex key pressed
stack containing new values
Alters: A,B,CC,X,U, Stack

This subroutine may be used for multiple 16 bit hex number input. Input format is the same as for the monitor tape read or write function keys, that is, entry of multiple parameters is controlled by the +/- keys, with prompts and default values for each parameter. Push the default values onto the stack before calling this routine, and retrieve the new values from the stack in the same order upon return. The following example should clarify the use of this subroutine.

```
LDD #DEFAULT3
PSHS D
LDD #DEFAULT2
PSHS D
LDD #DEFAULT1
PSHS D
LEAX PROMPTS,PCR
LDB #3              total number of inputs
```

```
                SWI3
                FCB $12                  call INPSEQ
                CMPA #$80                go key terminated?
                BNE ABORT
                CMPB #3                  got all three?
                BNE ERROR
                PULS D
                STD NEW1
                PULS D
                STD NEW2
                PULS D
                STD NEW3
                RTS
PROMPTS    FCB $40,$6D,$77,0             prompt for item 1
           FCB $40,$79,$77,0             prompt for item 2
           FCB $40,$5E,$77,0             prompt for item 3
```

Routine: VERSON
Call Number: $13
Address: ($F026)

Entry: –
Exit: R(A) = revision/version code
Alters: A,CC

This subroutine returns the revision code of CIEbug in the most significant digit (MSD) of R(A) and the version number in the least significant digit (LSD) of R(A). Current release of CIEbug is revision d, version 0, so R(A) would return with $d0.

Routine: TSTRAM
Call Number: $14
Address: ($F028)

Entry: –
Exit: R(CC) C bit = 0 if on-board RAM is ok,
R(CC) C bit = 1 if error detected in on-board RAM,
R(A) = IC number that produced the error
Alters: all

This routine provides a non-destructive RAM test for the first 4K of RAM (addresses $0000 – $0FFF). This RAM must exist in 2K segments. A ping-pong test is performed, and if the RAM passes, the highest location of RAM will be returned in R(Y). If an error is detected by this routine, the carry will be set and R(A) will contain either $03 or $04, the IC numbers of the on-board RAM in which the error was detected.

Routine: TSTROM
Call Number: $15
Address: ($F02A)

Entry: -
Exit: R(CC) C bit = 0 if ROM checksum = $00
R(CC) C bit = 1 if ROM checksum <> $00
R(A) = $02 (IC number of CIEbug ROM)
Alters: A,CC,X

Subroutine TSTROM may be used to verify the integrity of the CIEbug ROM. A checksum of the ROM contents is performed by this routine. CIEbug, as supplied, has a checksum of zero.

Routine: BMOVE
Call Number: $16
Address: ($F02C)

Entry: R(X) = start of block to be moved
R(Y) = end of block to be moved
R(U) = start of destination for block
Exit: block moved
Alters: X,Y,U,CC

A general purpose block move subroutine is provided by BMOVE. This subroutine is 'smart' and will move the block in such a way as to avoid damage to the data, so overlapping blocks may be accommodated. No other checking is done by this routine, so avoid moving into your own stack area or into CIEbug's variable area in the direct page.

Routine: FILL
Call Number: $17
Address: ($F02E)

Entry: -
Exit: -
Alters: none

This routine is one of the utility functions provided to enhance the functionality of your trainer. It is described in detail along with the monitor commands, in a previous section of this document.

Routine: RELADR
Call number: $18
Address: ($F030)

Entry: -
Exit: -
Alters: none

This routine is one of the utility functions provided to enhance the functionality of your trainer. It is described in detail along with the monitor commands, in a previous section of this document.

Routine: HEXCAL
Call number: $19
Address: ($F032)

Entry: -
Exit: -
Alters: none

This routine is one of the utility functions provided to enhance the functionality of your trainer. It is described in detail along with the monitor commands, in a previous section of this document.

Routine: FIND
Call Number: $1A
Address: ($F034)

Entry: -
Exit: -
Alters: none

This routine is one of the utility functions provided to enhance the functionality of your trainer. It is described in detail along with the monitor commands, in a previous section of this document.

Routine: DSPVER
Call Number: $1B
Address: ($F036)

Entry: -
Exit: after any key pressed
Alters: A,B,CC

This subroutine may be called to determine the current revision/version code of the CIEbug monitor. It displays the ROM revision/version code in the center of the display, and waits for any key to be pressed before returning.

Routine:	SADD
Call Number:	$1C
Address:	($F038)
Entry:	R(U) = parameter stack pointer, parameter stack containing Addend and Augend as signed two's complement words (16 bits each), i.e., 2,U = Addend ,U = Augend
Exit:	R(U) = R(U)+2, parameter stack contains sum i.e., ,U = Sum
Alters:	CC (OVFL) - ($D4) = FF if numeric overflow

Subroutine SADD performs signed two's complement addition on the top two words of a parameter stack pointed to by register U. One of the operands is removed from the stack, and the other is replaced by the sum. Direct page location OVFL ($D4) will be set to $FF if the sum exceeds the representable range for a 16 bit two's complement number.

Routine:	SSUB
Call Number:	$1D
Address:	($F03A)
Entry:	R(U) = parameter stack pointer, parameter stack containing Subtrahend and Minuend as signed two's complement words (16 bits each), i.e., 2,U = Subtrahend ,U = Minuend
Exit:	R(U) = R(U) +2 parameter stack contains difference i.e., ,U = Difference
Alters:	CC (OVFL) - ($D4) = FF if numeric overflow

Subroutine SSUB performs signed two's complement subtraction on the top two words of a parameter stack removed from the stack, and the other is replaced by the difference. Direct page location OVFL ($D4) will be set to $FF if the difference exceeds the representable range for a 16 bit two's complement number.

Routine:	SMUL
Call Number:	$1E
Address:	($F03C)
Entry:	R(U) = parameter stack pointer, parameter stack containing Multiplicand and Multiplier as signed two's complement words (16 bits each), i.e., 2,U = Multiplicand ,U = Multiplier
Exit:	R(U) = R(U)+2, parameter stack contains product, i.e., ,U = Product (MOD 65536)
Alters:	CC (OVFL) - ($D4) = FF if numeric overflow

Subroutine SMUL performs signed two's complement multiplication on the top two words of a parameter stack pointed to by register U. One of the operands is removed from the stack, and the other is replaced by the product. Direct page location OVFL ($D4) will be set to $FF if the difference exceeds the representable range for a 16 bit two's complement number.

Routine:	SDIV
Call Number:	$1F
Address:	($F03E)
Entry:	R(U) = parameter stack pointer, parameter stack containing Dividend and Divisor as signed two's complement words (16 bits each), i.e., 2,U = Dividend ,U = Divisor
Exit:	parameter stack contains quotient and remainder, i.e., 2,U = Quotient ,U = Remainder
Alters:	CC (OVFL) - ($D4) = FF if numberic overflow

Subroutine SDIV performs signed two's complement division on the top two words of a parameter stack pointed to by register U. One of the operands is replaced by the quotient and the other is replaced by the remainder. Direct page location OVFL ($D4) will be set to $FF if the difference exceeds the representable range for a 16 bit two's complement number.

Routine: SLOAD
Call Number: $20
Address: ($F040)

Entry: -
Exit: after EOF record
Parameters: (OFFSET) = load offset each record
(BITCON) = serial bit rate constant
Alters: all

Routine SLOAD is provided to read serial data in the form of ASCII 'S-Records', such as might be generated by a 6809 development system. Serial input is 7 data bits + 2 stop bits, at a rate determined by the direct page location (BITCON). Suggested values for a variety of standard bit rates are:

300 bps - 294 ($126)
600 bps - 145 ($91)
1200 bps - 70 ($46)
2400 bps - 33 ($21)
4800 bps - 14 ($0E)
9600 bps - 5 ($05)

Input data is to be applied to port B, bit 7 of the system PIA, the same port used in experiments. This signal should be at TTL levels, and referenced to the trainer power supply. Note that location (OFFSET) is a 16 bit offset which will be added to the load address present in each record to determine the effective load address at which the data will be deposited. As each serial character is received, it will be displayed in its hexidecimal representation on the right two digits of the display. SLOAD ignores header records (S0), and does not check the block check character at the end of each record. Data is deposited directly in the effective load address with no checking. Some caution should be exercised as to the destination, which can be controlled somewhat by a suitable selection of (OFFSET).

Routine: READ
Call Number: $21
Address: ($F042)

Entry: X = PIA Port B data register address
Parameters: (BITCON) determines bit rate.
see SLOAD above for values
Exit: A = seven bit character, right justified
Alters: A,CC

This subroutine may be used for general purpose serial input to the urainer. This routine will wait until a valid start bit is detected on port B, bit 7 (logic zero), after which a seven bit character is assembled into accumulator A. The validity of the stop bits are not checked. All this assumes the data direction register for port B has been programmed to make bit 7 an input, of course.

Routine: TYPE
Call Number: $22
Address: ($F044)

Entry: A = seven bit character to send,
X = PIA Port B data register address
Parameters: (BITCON) determines bit rate.
see SLOAD above for values
Exit: A = seven bit character with MSB set
Alters: A,CC

This subroutine may be used for general purpose serial ouput from the trainer. This routine will send out a start bit (logic 0), seven data bits as determined by the contents of accumulator A, and two stop bits (logic 1's). It is assumed by this routine that the PIA data direction register for port B has been programmed so as to make bit 0 an output.

Routine: DASH
Call Number: $23
Address: ($F046)

Entry: B = digit at which to display a dash
Exit: B = B + 1 (MOD 8)
A = pattern for dash on display
Alters: A,B,CC

Subroutine DASH simply displays a dash (segment g of the display) in a specified digit position.

Routine: DSPRD
Call Number: $24
Address: ($F048)

Entry: B = digit position to read (0-7)
Exit: A = pattern being displayed at that position
Alters: A,CC

A program may read the contents of the display by using this subroutine. Note that accumulator B specifies the digit position, and as such should be between 0 and 7 inclusive. The data returned in register A may require some interpretation, as it is the display pattern. For example, a '1' being displayed would read as $06. ($86 if decimal point lit).

Routine: DSPWR
Call Number: $25
Address: ($F04A)

Entry: A = data to write
B = digit position to write
Exit: -
Alters: CC

This subroutine is functionally identical to routine DSPPTN, with the exception that accumulator B is NOT incremented upon return. Bit definitions for the pattern remain the same as for that routine.

Routine: DSPBIN
Call Number: $26
Address: ($F94C)

Entry: B = data to display in binary
A = digit position to place decimal point (0-7). No point if > 7.
Exit: eight bits on display
Alters: A,B,CC

Routine DSPBIN is used to unpack and display the contents of accumulator B on the display as a sequence of eight ones and zeros, in MSB to LSB order. ($A5 would be displayed as 10100101.) A decimal point may be displayed in one of the digit positions, as detemined by the contents of accumulator A. If accumulator A is greater than 7, no decimal point will be displayed in any of the digit positions.

Routine: DSPTOS
Call Number: $27
Address: ($F04E)

Entry: U = pointer to a parameter stack with the value to be displayed at 0,U
(BASE) = radix to display in (2-16)
(SMODE) = 0 if unsigned display,
(SMODE) <> 0 if signed two's complement
Exit: number displayed
Alters: CC

A number may be displayed in an arbitrary radix by setting direct page parameters as desired and calling this routine. Either signed display or unsigned display is possible, so $FFFF may be displayed as either '-1' or '65535' (base 10). Several bytes on the hardware stack, as well as several bytes below the parameter stack are used by this routine, always following stack conventions.

Routine: NUMINP
Call Number: $28
Address: ($F050)

Entry: A = first number (=00 if none)
U = parameter stack pointer
Parameters: (BASE) = input radix
(SMODE) = 0 for unsigned input
(SMODE) = $FF for signed input
Exit: U = parameter stack with new entry
pushed on top (i.e., U=U-2)
A = non-number key that terminated input
Alters: A,B,CC,(SMODE)

Subroutine NUMINP allows the input of numbers in an arbitrary radix. For example, decimal numbers could be directly input. The number will continue to accumulate until a shifted numeric key is pressed, at which time the subroutine will return. There are three keys which are specially defined for this routine. The are: SHIFT /-, which acts as a clear entry , SHIFT /SET BR which toggles from signed display mode to unsigned display mode, and SHIFT /MOVE which negates the number being input (changes its sign from +/-).

Routine: ABSD
Call Number: $29
Address: ($F052)

Entry: D = (A:B) as a 16 bit two's complement
number
Exit: D = ABS(D)
Alters: D,CC

This routine may be used to take the absolute value of the 16 bit signed two's complement number in register D.

routine: NEGD
Call Number: $2A
Address: ($F054)

Entry: D = (A:B) as a 16 bit two's complement
number
Exit: D = $0000 - D
Alters: D,CC

This routine may be used to negate (change sign) of the 16 bit signed two's complement number in register D.

routine: NEGTOS
Call Number: $2B
Address: ($F056)

Entry: U = parameter stack pointing to a signed twos complement word on the top of the stack, i.e., at ,U.
Exit: U = parameter stack pointing at the negative of the previous number, i.e., (,U) = $0000 - (,U)
Alters: CC

Subroutine NEGTOS does for the top element of the parameter stack what subroutine NEGD does for the double accumulator. Note that a full word is involved here, so the parameter stack should contain a 16 bit number on top.

Routine: UMULT
Call Number: $2C
Address: ($F058)

Entry: U = parameter stack containing:
(2,U) = Multiplicand
(,U) = Multiplier
Exit: U = parameter stack containig:
(2,U) = low 16 bits of product
(,U) = high 16 bits of product
Alters: D,CC

UMULT may be used to multiply two unsigned 16 bit numbers to create a 32 bit product.

Routine: UDIV
Call Number: $2D
Address: ($F05A)

Entry: U = parameter stack containing
(2,U) = dividend
(,U) = divisor
Exit: U = parameter stack containing
(2,U) = quotient
(,U) = remainder
(OVFL) = $FF if divide by zero attempt
Alters: D,CC, (OVFL)

UDIV may be used to divide two 16 bit numbers producing a 16 bit quotient and a 16 bit remainder. Attempting to divide by zero will flush both quotient and remainder to $FFFF, and set direct page location (OVFL) to $FF. Other combinations may exceed the range of representable numbers, but this routine will not test for them.

Routine: DELAYH
Call Number: $2E
Address: ($F05C)

Entry: –
Parameters: (BITCON) = time delay constant
Exit: after 42 + 10*INT((BITCON/2) cycles
Alters: CC

A fixed time delay, approximately 1/2 of that provided by DELAYF (see below) is provided by this routine. The delay is given in terms of the processor cycle time, which is 4/3579545 seconds for an unmodified trainer.

Routine: DELAYF
Call Number: $2F
Address: ($F05E)

Entry: –
Parameters: (BITCON) = time delay constant
Exit: after 41 + 10*(BITCON) cycles
Alters: CC

A fixed time delay is provided by this subroutine. The delay is given in terms of the processor cycle time, which is 4/3579545 seconds for an unmodified trainer. A time delay constant of 0 is interpreted as 65536 counts.

Routine: ASCHEX
Call Number: $30
Address: ($F060)

Entry: A = seven bit ASCII digit or alpha character from 0-9, A-F.
Exit: A = binary value of input character
  = 00 if an invalid ASCII hexidecimal character.
Alters: A,CC

Subroutine ASCHEX may be used to convert a seven bit ASCII representation of a hexidecimal digit into its binary equivalent. For example, calling this routine with A=$41 would return A=$0A.

Routine: RING
Call Number: $31
Address: ($F062)

Entry: A = binary value
Exit: A = binary value rotated right one bit position,
with the former LSB now the MSB position.
Alters: A,CC

A 'ring shift right' operation is peroformed on the binary value in accumulator A by this routine.

Routine: FNDBIT
Call Number: $32
Address: ($F064)

Entry: B = bit number (0-7)
Exit: A = binary pattern for 2 exp(B)
i.e., a one in the bit position
specified by register B.
Alters: A,B,CC

Subroutine FNDBIT is useful for manipulating individual bits in a data structure. If register B contained 4 on entry, A would contain $10 (%00010000) upon exit. This value could be used in an ORA operation to set the given bit, or COMA then ANDA'ed to clear a particular bit.

Routine: TRFILE
Call Number: $33
Address: ($F066)

Entry: Hardware stack:
(4,S) = return address
(2,S) = load offset
(,S) = file name)
tape running, EAR connected
Parameters: (TAPESP) = tape speed parameter
Exit: after file load
Alters: all

Subroutine TRFILE may be used to read an object form tape into the trainer memory. A tape of the proper format may be written out using the trainer WRITE command, or the subroutine TWFILE below. Such a tape contains file information, including load address information. You may choose to offset the normal load address by setting parameters in the hardware stack (S). Note that this subroutine makes use of parameters pushed onto the stack, so it may not be called as a 'normal' subroutine. The follwing example illustrates a method for calling this subroutine:

```
        LEAX RETN,PC     SET RETURN ADDRESS
        PSHS X
        LDD #OFFSET
        PSHS D           LOAD OFFSET
        LDD #FNAME
        PSHS D           FILE NAME IN 3LSD
        JMP ($F066)      CALL TRFILE
   RETN NOP              OR OTHER CODE UPON RETURN
```

Routine CLOADA
Call Number: $34
Address: ($F068)

Entry: tape up to speed and synchronized
Y = PIA data register for port A
Parameters: (TAPESP) = tape speed parameter
Exit: A = byte read from tape
R(CC) C bit = 1 if parity error

This subroutine may be used to read a single byte from the tape. Data format is start bit (logic zero-1200 HZ), eight data bits (LSB first) and even parity.

Routine: CLDCY
Call Number: $35
Address: ($F06A)

Entry: Y = PIA data register for port A
tape up to speed and synced
Parameters: (TAPESP) = tape speed parameter
Exit: R(CC) C bit = bit value read from tape
Alters: A,B,CC

This subroutine may be used to distinguish between a 2400 HZ tone and a 1200 HZ tone being input into the EAR jack. A 2400 HZ tone is recognized as a logic 1, while a 1200 HZ tone is represented as a logic zero. Incoming data will be echoed to the internal speaker.

Routine: TWFILE
Call Number: $36
Address: ($F06C)

Entry: Hardware stack:
(6,S) = return address
(4,S) = end address of block to write
(2,S) = start address of block to write
(,S) = file name
tape running, MIC connected

Exit: after file written
Alters: all

Subroutine TWFILE may be used to prepare a tape file from data resident in trainer memory. A leader is written out (10 seconds of 2400 HZ), then a short name file block, followed by the data, finishing with an overall checksum. This form is compatable with the monitor READ function, or the TRFILE subroutine above. Note this subroutine makes use of parameters pushed onto the stack, so it may not be called as a 'normal' subroutine. The following example illustrates a method for calling this subroutine.

```
     LEAX RETN,PC     SET RETURN ADDRESS
     PSHS X
     LEAX EA,PC       MARK END OF BLOCK
     PSHS X
     LEAX SA,PC       MARK START OF BLOCK
     PSHS X
     LDD #FNAME       SET FILE NAME
     PSHS D
     JMP ($F06C)      CALL TWFILE
RETN NOP              OR OTHER CODE UPON RETURN
```

Routine:	CSENDA
Call Number:	$37
Address:	($F06E)
Entry:	tape up to speed and synchronized Y = PIA data register for port A U = sine table for waveform synthesis A = byte to send to tape
Exit:	after byte sent
Alters:	A,B,CC

This subroutine may be used to write a single byte to the tape. Data format is start bit (logic zero - 1200 HZ), eight data bits (LSB first) and even parity.

Routine:	CSNDCY
Call Number:	$38
Address:	($F070)
Entry:	Y = PIA data register for port A U = 16 byte sine table for waveform synthesis. tape up to speed and synced R(CC) C bit = bit value to write
Exit:	after bit written
Alters:	A,B,CC

This subroutine may be used to write either a 2400 HZ or 1200 HZ tone to the MIC jack as a sinusoid. A 2400 HZ tone is sent for CY=1, and 1200 HZ tone is sent for CY=0 on entry.

Routine: JMPCMD
Call Number: $39
Address: ($F072)

Entry: A = command code
B = $0E
Exit: upon return from command subroutine
Alter: all

Subroutine JMPCMD may be used to obtain monitor functions from within a program. The command code is the key matrix coordinate of the monitor function key desired + $80 (optional for some functions). So to invoke the memory display/modify function, call this routine with register A = $11 if data entry, $91 if address entry desired. Commands READ, WRITE, STEP, and GO are screened out by this subroutine.

Routine: Reserved 0 - Reserved 3
Call Numberer: $3A - $3D
Address: ($F074-$F07B)

Entry: none
Exit: -
Alters: -

These entries in the jump table are reserved for future expansion of CIEbug. Calling these routines will result in unpredictable results.

Routine: SINTAB
Call Number: (none)
Address: ($F07C)
Entry: none
Exit: -
Alters: -

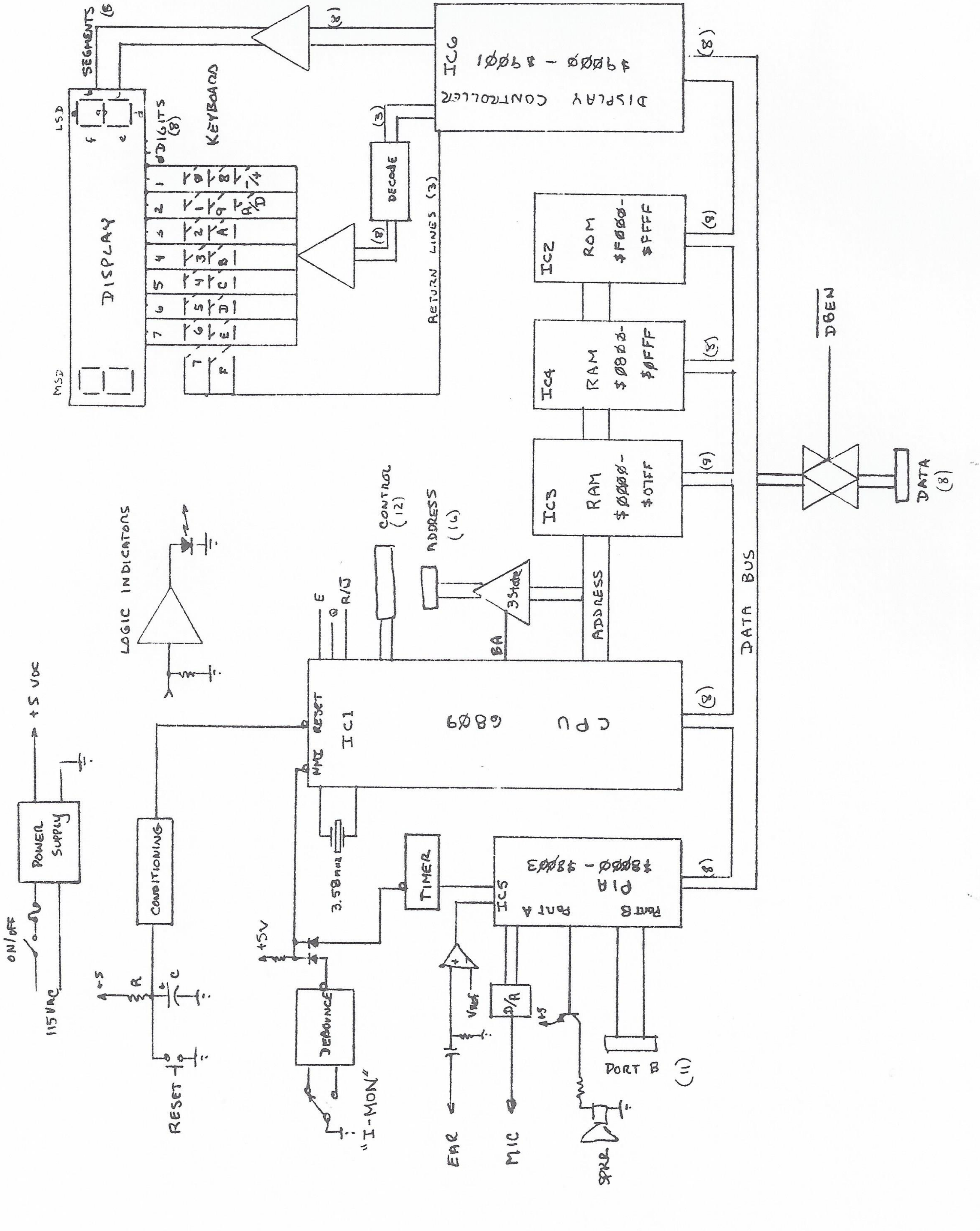
This entry in the jump table is NOT executable. It simply gives the address of the sine table used for synthesis of the cassette waveform. A typical use of this enty would be LDX $F07C, so that X could now be used to point to the data table inside the CIEbug ROM.

Routine: SEGTAB
Call Number: (none)
Address: ($F07E)
Entry: none
Exit: -
Alters: -

This entry in the jump table is NOT executable. It simply gives the address of the hex digit to seven segment display pattern table used by CIEbug to display hex digits. A typical use of this entry would be LDX $F07E, so that X could now be used to point to the data table inside the CIEbug ROM.

VI - TRAINER HARDWARE INFORMATION

SYSTEM BLOCK DIAGRAM


SEGMENTS (8)
(8)
IC6
$9000 - $9001
DISPLAY CONTROLLER
(8)
LSD
DIGITS (8)
KEYBOARD
DISPLAY
MSD
(3)
DECODE
(8)
RETURN LINES (3)
IC2
ROM
$F000-
$FFFF
(9)
IC4
RAM
$0800-
$0FFF
(8)
IC3
RAM
$0000-
$07FF
(9)
DBEN
DATA
(8)
DATA BUS
CONTROL
(12)
ADDRESS
(16)
3State
ADDRESS
BA
E
Q
R/W
LOGIC INDICATORS
+5 VDC
POWER SUPPLY
ON/OFF
115VAC
CONDITIONING
RESET
R
C
+5
NMI
RESET
IC1
6809
CPU
(8)
3.58MHz
TIMER
IC5
PIA
$8000 - $8003
Port A
Port B
(8)
+5V
DEBOUNCE
"I-MON"
EAR
MIC
Vref
D/A
+5
PORT B
(11)
SPKR

CONNECTORS

Student Blocks:
0.025" sq posts on 0.1" centers.
Elenco Electronics Inc. #9905, available through the CIE Bookstore.

Card Edge Expansion:
Dual 25 pin PCB edge, 0.1" spacing.
Texas Instructments Inc. #C5-25 (solder), #C6-25 (wire wrap)
AP Products Inc. and Robinson Nugent Inc. make a variety of ribbon cable connectors and assemblies that are compatable with the Trainer Expansion connector.
Radio Shack #276-1566, #276-1545 can also be used.

Card Edge Connector Pin Definition:

```
  Bottom          Back View      Top

       +--       50--] [--49    --+
       :         48--] [--47      :
       :         46--] [--45      :
  GND--+         44--] [--43     +5 vdc
       :         42--] [--41      :
       :         40--] [--39      :
       +--       38--] [--37    --+
  FIRQ*          36--] [--35     RESET*
  R/W*           34--] [--33     DMA*
  HALT*          32--] [--31     MRDY
  IRQ*           30--] [--29     BS
  BA             28--] [--27     Q
  E              26--] [--25     DBEN*
  D7             24--] [--23     D6
  D5             22--] [--21     D4
  D3             20--] [--19     D2
  D1             18--] [--17     D0
  A15            16--] [--15     A14
  A13            14--] [--13     A12
  A11            12--] [--11     A10
  A9             10--] [-- 9     A8
  A7              8--] [-- 7     A6
  A5              6--] [-- 5     A4
  A3              4--] [-- 3     A2
  A1              2--] [-- 1     A0

  Bottom          Back View      Top
```

Where a '*' is used instead of an overbar to indicate a signal is active low.

Power Supply:

Dual Banana Plugs, 3/4" spacing.
Pomona MDP series dual plugs,
Radio Shack #274-721, #274-730 are single plugs that
are compatable with the Trainer power supply binding posts.

SIGNAL SPECIFICATIONS AND MAXIMUM LOADING:

Power Supply:
+5 Vdc +/- 0.25 volt @ 250 mA Maximum

Student Logic Indicators:

Input Impedance:	10K ohms typical
Input Current:	0.5 mA typical at Vin=+5 vdc
Logic Threshold:	> 2.5 vdc = LED on < 2.5 vdc = LED off (typical values)

Student Blocks, Expansion Connector:

Internally, the signals on the card edge connector and the corresponding signals on the student blocks are connected together. Therefore the following specifications apply to either point, and the total load on a given signal must not exceed the specified values.

Logic Levels	Voh, Vih	>2.5 volts
	Vol, Vil	<0.8 volts

Fanout	A0-A15,D0-D7,E,Q,BA,BS Four LSTTL equivalent loads minimum R/W*,RESET* Two LSTTL equivalent loads
Fanin	DBEN*,IRQ*,MRDY,HALT*,DMA*,FIRQ* One LSTTL equivalent load + 10K pullup (approximately two LSTTL loads)

CABLE

Cassette:
Dual 1/8" Mini phone plug to 1/8" Mini phone plug.
Cables are available from the CIE Bookstore. Two would
be needed, one for MIC, the other for EAR.

SEMICONDUCTOR PARTS LIST

Designation	Generic or Manufacturers Part Number
IC1	MC6809P CPU
IC2	CIEbug ROM (available from CIE)
IC3	6116 2K X 8 STATIC RAM
IC4	6116 2K X 8 STATIC RAM (optional)
IC5	MC6821P or SY6521 PIA
IC6	8279 KEY/DISPLAY CONTROL
IC7	74LS00 QUAD 2-INPUT NAND
IC8	74LS139 DUAL 2-4 DECODER
IC9	74LS138 3-8 DECODER
IC10	CA3160E OPAMP
IC11	CD40103 BINARY COUNTER
IC12	74C244 OCTAL BUFFER
IC13	74C244 OCTAL BUFFER
IC14	74LS245 OCTAL TRANCEIVER
IC15	74C244 OCTAL BUFFER
IC16	4503 HEX BUFFER
IC17	4050 HEX BUFFER
IC18	4050 HEX BUFFER
IC19	4093 QUAD NAND SCHMIDT TRIGGER
IC20	4051 3-8 MUX
IC21	DS8654 DISPLAY DRIVER
IC22	DS8863 DISPLAY DRIVER
----	7805 REGULATOR

DISPLAYS

7-segment LED's	MAN6740

CIEBUG MONITOR DIAGNOSTICS

Because the microprocessor is capable of testing some of the components of the training laboratory under program control, the CIEbug monitor includes resident diagnostic subroutines which may be used to insure the validity of the laboratory. No action is needed by the operator. Each time the trainer first receives power, a series of internal tests are performed. These include tests of the CPU itself, the ROM containing the CIEbug monitor, the system side of the PIA, the keyboard/display controller, the sound output bit, the speaker, and the on-board RAM. This series of tests is only done during a cold start sequence, and the results can be observed as the initial display pattern and the short tune that is played over the speaker.

If an error is detected by the system test routines, the display will attempt to indicate the word 'Error' and the IC number of the component most likely the cause of the error. For example, if the RAM test found a bad bit in the expansion RAM (IC4), the display would indicate 'Error 04', since IC4 is the expansion RAM designation. This error display will be punctuated by a repreating short beep, and the system will stay in this state until power is removed or the reset button is pressed.

IN CASE OF TROUBLE...

The CIE trainer was designed to serve as a reliable training aid. It uses many high-technology components which are internally very complex, but perform simple external actions, and which normally have a long life. In the unlikely event that you experience problems with the training laboratory, consult the appropriate section of the manual, and if that fails, contact CIE. Certain failures may be the result of simply misunderstanding what the trainer is supposed to do. Some components are socketed, but it is not recommended that you replace components first, since the training laboratory is thoroughly tested before shipment.

CASSETTE TAPE RECORDER GUIDLINES

This section contains some recommendations concerning the use of a cassette tape recorder with the MTL-1 microprocessor trainer.

1) Use a moderate quality audio cassete tape recorder. Inexpensive recorders, particularly those with a DC bias/erase head are not recommended. Moderately priced audio tape recorders that have been tested and approved by CIE for use with the microprocessor trainer are available from the CIE Bookstore.

2) Use high quality audio tapes to save your programs. Cassette tapes specifically designated for computer data storage (using an audio cassette recorder) may also be used. Tapes should have good high frequency response and be free from "drop outs". Audio cassette tapes approved for use with the trainer are available from the CIE Bookstore.

3) Keep the recorder tape transport path clean and degaussed (heads and pinch rollers). Head alignment is essential to tape interchangeability.

4) Whenever possible, operate the recorder using an AC adapter. Batteries, if you must use them, should be replaced often.

5) Playback volume control should be set between 1/4 and 3/4 of maximum. Tone control, if present, should be set at its highest setting. Some adjustment may be made by listening to the data echoed over the trainer's internal speaker during a tape read operation. Adjust settings for a "clear" tone during playback.

6) Position the tape past any leader before recording. Make sure that you have the MIC cable connected to the recorder. You will hear data from the trainer's internal speaker even if the recorder is not connected. Some recorders require that you record with the EAR plug disconnected.

7) Keep a directory of the contents of each tape, including such information as file name, load address, ending address, and a short description of the program or data. Make sure this information is updated as the tape is used.

8) Verify each file after you record it. Rewind the tape and use the tape READ command of CIEbug to search for the file just written, with a load offset such that the file will be loaded into some unoccupied available RAM. Then even if an error does occur, you won't lose the original copy of your data. Alternatively, use the tape READ command to search for a file that does NOT exist on the tape. This will inhibit CIEbug from altering anything in memory, but each file encountered will be checked for parity and checksum errors exactly as if it were being loaded into memory.

9) Rewind cassettes before removing them from the recorder. Store cassettes in their cases, away from heat or magnetic fields.

10) Should you still experience problems after following the above recommendations, try running the speed parameter adjustment programs given in the next section.

CASSETTE SPEED PARAMETER ADJUSTMENT

While the cassette tape routines built into CIEbug were designed to work with a number of tape recorders, some combinations of trainers and recorders may experience problems. For this reason, an operator-alterable tape speed parameter was incorporated into the tape read routines. This parameter can be used to compensate for minor speed variations during playback. A default value ($16) is set up (in location TAPESP - $EA on the monitor direct page) by CIEbug when the trainer first receives power. In order to check this parameter, and adapt it to your particular trainer and cassette recorder, enter the following program beginning at address $0000. The left column gives the hex address, and the next eight columns give the data to be placed in consecutive locations.

```
Addr
----
0000  34  57  11  3F  02  8E  00  00
0008  FE  F0  7C  C6  10  A6  C0  b7
0010  80  00  8d  09  5A  26  F6  30
0018  1F  26  Ed  35  d7  1A  00  39
0020  34  17  11  3F  02  E6  84  2b
0028  FC  63  84  E6  84  2A  FC  63
0030  84  4F  E6  84  2A  03  4C  26
0038  F9  63  84  9b  EA  47  89  00
0040  97  EA  5F  11  3F  05  11  3F
0048  0C  27  E0  35  97  00  00  00
```

Connect the MIC cable to the cassette recorder and trainer, insert a blank tape, set record mode, and run the above program (using the GO command of CIEbug) beginning at location $0000. This will write out a reference tone to the tape for approximately 36 seconds. After the program returns to CIEbug, stop the tape, disconnect the MIC cable, and connect the EAR cable to the trainer and recorder.

Rewind the tape, set play mode, and run the above program beginning at location $0020. This part of the program uses the reference tone on the tape to calculate a new tape speed parameter, which is deposited into the proper location on the monitor direct page, and displayed in the rightmost two digits of the display. Run the tape for approximately 10-15 seconds, adjust the tone and volume controls to obtain the most "constant" reading of the display, then press any unshifted key on the keypad. This will return control to CIEbug. Stop the tape, rewind and save it for future use. Also write down the tape speed parameter that you last saw on the display. You will have to enter that value into location TAPESP each time you turn on the trainer, or anytime the trainer does a "cold" start.

The above program will set the proper value for you each time it is run. Representative values are in the range $10 to $1C with the nominal value being $16. If the value calculated by the program is significantly different ($00 or $FF for example), or if the program appears to "hang up" with no display, you may have a problem withe the trainer hardware.

VII - 6809 MICROPROCESSOR REFERENCES

Listed below are a few selected publications which might prove useful to students, programmers, and experimenters using the 6809 based CIE Microprocessor Training Laboratory.

CLEVELAND INSTITUTE OF ELECTRONICS LESSONS

The following publications are lessons that are included in various CIE career, industrial, and associate degrees courses. Consult the most recent CIE course catalog for additional information.

Lesson 5544	Introduction to Computers and Microprocessors
Lesson 5376	Registers
Lesson 5378	Arithmetic Logic Units
Lesson 5380	Timing and Control
Lesson 5382	Memory Units
Lesson 6702	Using the CIE 6809 Microprocessor Trainer
Lesson 6704	Number Systems and Data Codes
Lesson 6706	Digital Arithmetic and Logic Operations
Lesson 6708	The 6809 Architecture and Operation
Lesson 6710	Programming I - Straight Line
Lesson 6712	Programming II - Branches and Jumps
Lesson 6714	Programming III - Stacks and Subroutines
Lesson 6716	Programming IV - I/O and Interrupts
Lesson 6718	Program Design and Testing: Projects
Lesson 6720	Semiconductor Memory Interfacing
Lesson 6722	Elementary I/O and the PIA
Lesson 6724	Output- Displays and Sounds
Lesson 6726	Inputs from Switches and Keyboards
Lesson 6728	Interrupt Systems
Lesson 6730	Serial I/O: UART/ACIA
Lesson 6732	Analog I/O: D/A and A/D Conversion

OTHER PUBLICATIONS

The following books and reference publications are available through most technical book stores and many are available from the CIE BOOKSTORE

Microprocessors Data Manual (DL120)
Motorola Semiconductor Products Inc.

MC6809 Programming Manual (MC6809PM-AD)
Motorola Semiconductor Products Inc.

MC6809 - MC6809E 8-bit Microprocessor Reference Card
Motorola Inc. - MOS Integrated Circuits Division

6809 Assembly Language Programming
by Lance A. Leventhal
Osborne/McGraw-Hill

The MC6809 Cookbook
by Carl D. Warren
TAB Books Inc.

Programming Microprocessor Interfaces for Control and Instrumentation
by Michael Andrews
Prentice-Hall, Inc.

Programming the 6809
SYBEX Inc

Microprocessor Interfacing Techniques
by Austin Lesea and Rodnay Zaks
SYBEX Inc

An Introduction to Microcomputers:
Volume I - Basic Concepts
by Adam Osborne
Osborne/McGraw-
(general)

Microprocessor Fundamentals
by Roger L Tokheim
Schaum's Outline Series- Mc Graw Hill Book Company
(general)

Introduction to Microcomputing
by Sydney B. Newell
Harper & Row
(6800)

Introduction to Microcomputer-Based Digital Systems
by James W. Gault and Russel L. Pimmel
McGraw-Hill Book Company
(6800)

The original manual had additional pages, consisting of datasheets. They are not reproduced here, because they are widely available online:

Motorola MC6809, MC68A09, MC68B09 datasheet: pages 60 – 91

Motorola MC6821, MC68A21, MC68B21 datasheet: pages 92 – 101

Intel 8279, 8279-5 datasheet: pages 102 - 113